Workshop Proceedings of the 10th International Conference on Intelligent Environments
J.C. Augusto and T. Zhang (Eds.)

doi:10.3233/978-1-61499-411-4-171

Design for Failure: Intelligent Systems Learning from Their Mistakes

Davy PREUVENEERS [a,1], Ching Yu CHEN [b], Arun RAMAKRISHNAN [a], Ming-Whei FENG [b], Ping-Feng WANG [b], Yolande BERBERS [a]

[a] *iMinds-DistriNet, Celestijnenlaan 200A, B-3001, Heverlee, Belgium*
{ *davy.preuveneers, arun.ramakrishnan, yolande.berbers* }*@cs.kuleuven.be*
[b] *Institute for Information Industry, 4 floor, No.87, Lianyun St., Zhongzheng Dist., Taipei City 100, Taiwan (R.O.C)*
{ *chingyuchen, mfeng, pfwang* }*@iii.org.tw*

Abstract. Smart environments aim to make the life of their inhabitants more comfortable by having context-aware systems continuously work together to assist people with their daily tasks. However, all too often these assistive technologies are naively or optimistically developed assuming that systems can always anticipate what users want. Furthermore, the more these smart systems grow in complexity, the more prone to failure they become. The overall goal of this paper is to define new concepts and methodologies for the development of more reliable smart applications, and propose middleware support to analyze failures in context-aware behavior, culminating in a software-based safeguard that improves robustness against unforeseen human interventions, exceptional circumstances and unexpected events.

Keywords. Intelligent systems, design for failure, context, learning

Introduction

Intelligent systems continue to grow in size and complexity. For example, the Airbus A380 has about 250000 sensors and parameters that continuously monitor the airplane, and advanced computer systems that automate many flying tasks and advice pilots. However, this advice may not always be helpful or can in some cases be incorrect, so a healthy skepticism by the pilots is always required.

While state-of-the-art jumbo jet airplanes are definitely examples of smart environments, the vision of ubiquitous computing (ubicomp) in our everyday's surroundings as described in Mark Weiser's seminal article has still not become reality for similar reasons of preventing inappropriate behavior. Many middleware solutions have been proposed in the last decade for developing ubiquitous computing applications. They can successfully deal with the heterogeneity in sensor and actuator networks [1], simplify the portability of smart applications [2] to other

[1] Corresponding Author.

hardware and software platforms, and facilitate the development of context-aware distributed applications [3]. With events and complex event patterns [4] providing a system's abstraction for interactions between a user and systems in a smart environment, new algorithms are proposed to discover frequent relationships between sensor data and actions carried out by the user [5,6,7,8] to learn common behavior and help intelligent environments to act proactively, anticipating the user's needs and preferences [9]. These building blocks provide the foundations for new assistive technologies and emerging paradigms like Ambient Assisted Living, intelligent autonomous vehicles, e-health, etc.

However, the complexity of integrating computing in our daily lives and making our environment intelligent, proactive and intuitive to use, has proven to be very high. An important challenge is managing and mitigating the increasing potential for failure in a smart environment. As these ubicomp components interact with each other in sophisticated ways, and as they grow in numbers, so does the complexity which will adversely impact the reliability [10] of the smart environment. Recent work [11] argues for model checking approaches to improve the reliability and safety of such environments [12] against unexpected circumstances that would otherwise lead to failures.

In this work, we aim for a first step towards a more systematic method to analyze inconsistencies in context-aware rule-based behavior at runtime to be more robust against unforeseen human interventions, exceptional circumstances and unexpected events. Faced with these observations, we try to provide an answer to the following questions:

1. How can we detect inconsistencies in context-aware decisions and actions that drive the dynamic behavior of smart systems?
2. How can learning of patterns in event streams be used to anticipate and prevent failures?
3. How can context-aware event-based interaction patterns be used to identify significant deviations from common or expected human interactions?
4. How can we embed a software safeguard in the design of the application to reduce the risk of failures during unexpected contextual circumstances?

In section 1, we introduce robustness and design for failure as two important concepts for smart systems. Section 2 presents an intelligent environment use case driven-by context-aware behavior rules and section 3 presents a methodology to detect deviating behavior. In section 4 we will elaborate on some preliminary qualitative results with our approach using our SAMURAI framework. We conclude and propose further work in section 5.

1. Robustness and design for failure

Robustness [13] is a characteristic of a system that ensures it performs well not only under ordinary conditions but also under unusual conditions that stress the developers' assumptions of the system. Hence, it is difficult for a developer to discover and eliminate all errors when the application will be active outside a restricted or limited context. As a result, fragile context-aware applications are

subject to subtle or more severe errors that only make their presence known in unusual circumstances. This problem is aggravated when users can customize the contextual triggers (event-condition-action rules) that modify the behavior of a ubicomp system, causing inconsistencies among these rules (e.g. turning on the heating and the air conditioning at the same time).

Design for failure [14] is a methodology that - rather than always trying to prevent any type of failure - aims to be prepared for failure. With a plan for what to do when a system fails, we can aim to minimize the impact of a failure. Smart systems that can tolerate partial failures, deal with contextual inconsistencies (e.g. an erroneous sensor reading) or improper human handling, and minimize the impact of a failure through graceful degradation will improve the robustness and the quality of experience for the end-users. However, the kind of specification models and testing and verification techniques that we need for developing smart and reliable environments is still an open area of research.

2. Internet-of-Things use case of a smart building

We are tackling these challenges within the frame of the FP7 BUTLER project[2]. The objective of BUTLER is the creation of a horizontal IoT platform supporting several domains of our daily lives − including home, health, smart cities, energy, transport, shopping, etc. − all at once.

The scenario that we will use to motivate and demonstrate our approach targets a real-life integrated Internet-of-Things (IoT) smart living application deployed on the 10th floor of the ITeS building (see Figure 1), involving an internet of sensors, actuators and people. The main purpose of the intelligent IoT system and interactive applications in this commercial building is to balance three objectives, i.e. operation efficiency, comfortableness and energy bill savings.

Energy savings can be directly measured by monitoring the total power consumption and estimating the electric bill based on the various tariffs selected for the building. The comfortableness can be measured with the Predicted Mean Vote (PMV) and Predicted Percentage Dissatisfied (PPD) [15] measures, based on the spatial calculation of temperature and humidity. In fact, the PMV is the average score of a large group of people on a seven-point thermal sensation scale (+3=hot, 0=neutral, −3=cold). The value can be computed based on the metabolic rate of the individual, clothing insulation, air temperature, mean radiant temperature, air velocity and humidity. The PPD − a function of the PMV − is a value representing the percentage of thermally dissatisfied people who feel too hot or too cold. The operation efficiency will be activity dependent and is fairly ambiguous to measure (and is out of scope for this work).

The environment is setup with a few hundreds of sensors of different types that will be responsible for collecting data from the global environment:

[2]http://www.iot-butler.eu/

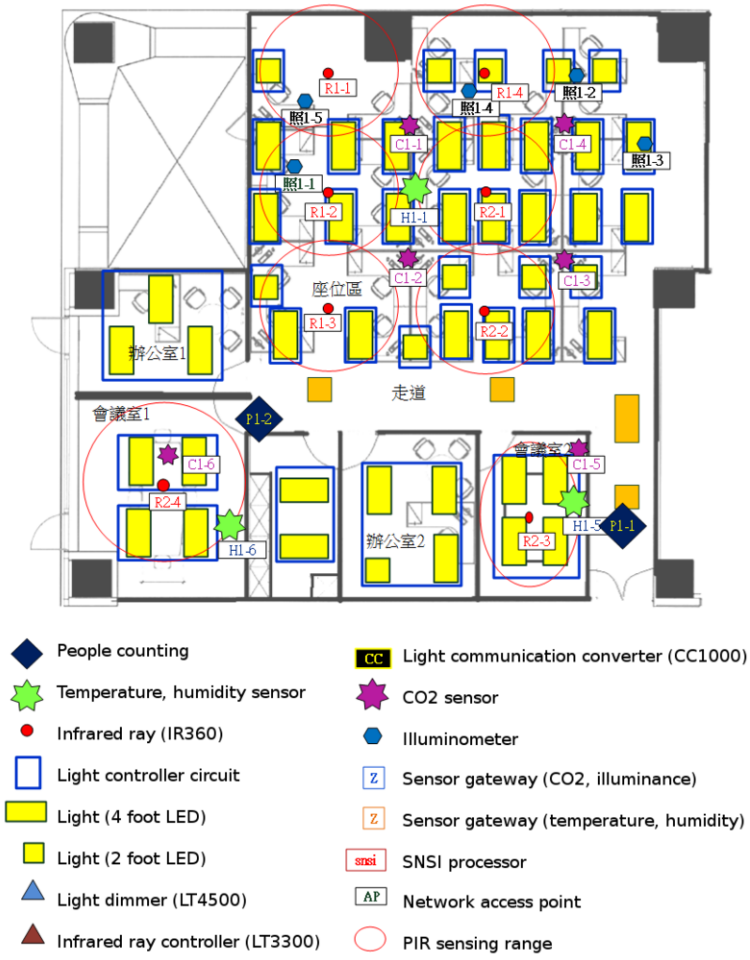

Figure 1. The ITeS integrated smart living use case

- Air condition
- CO_2 sensor
- Illumination sensor
- Infrared ray
- Light
- People counter
- Power meter
- Temperature
- Humidity
- ...

The building is also equipped with a variety of actuators that can manipulate the environment. These include chillers, cooling towers, ice tanks, pumps, lightning, sun shields, etc. Individuals can set their comfort preferences. The HVAC system, however, is partially managed centrally for the whole building following a fixed program, but additional air conditioning units are available on this floor that can be configured locally and individually. In general, we can describe the dynamic behavior of this floor with context-aware decision and adaptation rules (following the typical *event-condition-action* paradigm).

$$\textbf{if } (\text{time} = 8{:}00 \text{ AM}) \textbf{ then } \text{central\_airco} = \textit{on} \tag{1}$$

$$\textbf{if } (\text{people\_count} = 0) \textbf{ then } \text{light} = \textit{off} \tag{2}$$

$$\textbf{if } (\text{temperature} \geq 24^{\circ} C) \textbf{ then } \text{fanspeed} = \textit{high} \tag{3}$$

$$\textbf{if } (\text{illuminance} < 500 \ \textit{lux}) \textbf{ then } \text{light} = \textit{on} \tag{4}$$

$$\textbf{if } (\text{illuminance} < 500 \ \textit{lux}) \textbf{ then } \text{sun\_shields} = \textit{open} \tag{5}$$

$$\textbf{if } (\text{motion} = \textit{false}) \textbf{ then } \text{fanspeed} = \textit{low} \tag{6}$$

This is merely a subset and a simplified representation of the rules that might drive the behavior of an automated system. The complexity of ensuring robustness is caused by the fact that many features may interact with one another. For example, closing the sun shields does not only affect the temperature inside, but also the luminosity. Inconsistencies may emerge due to different context triggers, as illustrated in rules (3) and (6).

Certain rules might trigger actuators to reach the correct or desired state, but do this in a less than optimal way. As shown in rules (4) and (5), both the lights and sun shields can affect the luminosity in a building to reach a certain level of comfortableness. However, from an energy savings bill, it might be cheaper to open the sun shields rather than turning on the lights everywhere.

In previous work [16], we discussed consistency in context-aware behavior with a model checking approach using the SPIN tool. From a pragmatic point of view, however, it is hard to get a formal model that accurately mimics the behavior of the environment in response to stimuli. Individual preferences may change over time (e.g. depending on the season). Certain actions cause an immediate and local effect on the environment (e.g. a light switch), whereas others (e.g. the air conditioning) take a certain amount of time to reach the desired objective, and this duration may depend on the activity of other units in the vicinity. To complicate things even more, the air conditioning units cool the environment using ice banks and air pumps, with the ice being produced the night before when electricity is cheaper. As such, finding a balance between comfort and energy savings is not trivial from an optimization perspective.

3. Towards a software-based safeguard for rule-based behavior at runtime

In general, an *event-condition-action* (ECA) rule that models context-aware behavior states that if a particular situation emerges, a predefined action should be carried out. Such rules do not explicitly state the overall objective. For example, rule (3) defines a trigger for increasing the fan speed of the air conditioning unit, but does not state the desired affect, e.g. a temperature decrease of 2°C over the next 20 minutes. There are a few concerns with such rule based behavior:

1. The desired effect or the post-condition of a rule is not explicitly stated.
2. The impact of the action to reach the desired effect cannot be measured.
3. The effect can not always be attributed to the action of a given rule.

3.1. Enhancing event-condition-action rules for adaptive behavior

While specifying rules to drive the dynamic behavior of an intelligent environment is rather intuitive, the aforementioned observations give rise to robustness concerns against unforeseen human interventions, exceptional circumstances and unexpected events. We therefore extend the notion of ECA rule-based behavior with two additional concepts: *utility* and *effect*. We actually build upon *design-by-contract* principles in software engineering:

- **pre-condition**: a predicate that must be true just prior to the execution of a method. It indicates the assumption that certain things are true about the environment.
- **post-condition**: a predicate that must always be true just after the execution of a method. It indicates the changes made to the state of the world by invoking the method.

In our work on rule-based adaptive and intelligent systems, the method to be executed is the *action* part of the *event-condition-action* rule:

$$\textit{events}: \ \textbf{if}\ (\textit{conditions})\ \textbf{then}\ \ \textit{action}\ \ \textbf{ensure}\ \ \textit{effect}\ [\textit{utility}] \tag{7}$$

The *condition* defines a pre-condition on the *events* for the *action* of this rule to be carried out. Note that even though the condition of an event-condition-action rule is true, this does not mean that the corresponding adaptation will always be executed. The predicate states a pre-condition, but no obligation to execute. This gives us the opportunity to choose among multiple matching rules that might achieve the same desired effect, but with a different outcome on the optimization criteria, i.e. saving energy and increasing the comfort levels. For an example, see rules (4) and (5).

The *effect* defines the desired post-condition after the rule was executed. Contrary to design-by-contract best practices, we do not and cannot guarantee that the post-condition will always hold after executing the action due to the presence of interfering components.

The *utility* is a value in the range $(-\infty, 1.0]$ that measures how well the resulting effect reaches the desired situation relative to the previous situation. If the rule has an overall low but positive utility, then the rule is not effective. If the utility is negative, the rule is counter productive and worsening the situation.

For representing the desired effect, we use the following notation to state that a certain effect e holds after executing an action a:

- `e`: e has to hold.
- `a ensure e`: e has to hold after executing the action a.
- `<t>e`: e has to hold after a time t.
- `<t,p>e`: e has to hold after a time t for at least a period p.
- `e U f`: e has to hold until at least property f holds.

In this representation, `<0>e` and `<0,inf>e` are equivalent to `e`. We can then state the desired effect of an event-condition-action rule as follows:

$$\textit{temp}: \textbf{if}\ (\text{temp} \geq 24)\ \textbf{then}\ \text{fanspeed} = \textit{high}\ \textbf{ensure}\ \ {<}\,300\,{>}\,(\text{temp} < 24) \tag{8}$$

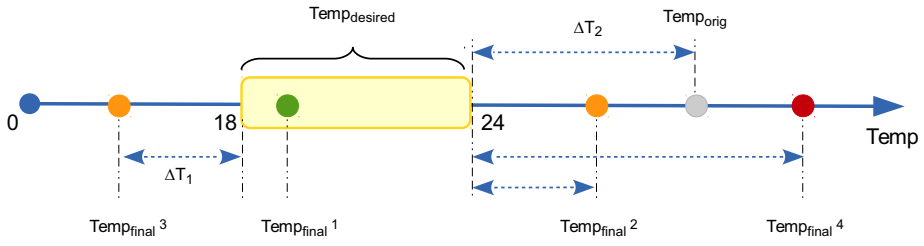

Figure 2. Measuring the utility of a rule

This rule states that if the temperature goes above $24°C$, that the fan must be set at high speed, and that the desired effect is a temperature below $24°C$ after 5 minutes. If the desired effect (i.e. $temp < 24$) is not specified explicitly, we assume the negation of the rule condition is the intended desired effect (as is the case in the above rule).

Note that a high fan speed could cause a low temperature of $15°C$. This is a valid temperature according to the above rule. However, at this stage we do not want another rule to activate the heating to increase the temperature again from $15°C$ to about $21°C$. It is therefore recommended to bound the desired state:

$$temp : \textbf{if} \ldots \textbf{ensure} \ < 300 > (\text{temp} < 24 \ \&\& \ \text{temp} > 18) \tag{9}$$

The safety properties of an intelligent system should express that something bad never happens independent of which events occur and what action is taken. If this bad behavior is represented as *bad*, we can notate this as:

$$events : \ \textbf{ensure} \ !\text{bad} \tag{10}$$

3.2. Computing the utility and optimality of a rule

Each event-condition-action rule defines the desired effect after the action has been carried out. To compute the utility of this rule, we define a metric that measures the distance between the *original* state, the *desired* state and the *final* state. We illustrate this in Figure 2 and by reusing the temperature example:

$$utility = 1 - \left| \frac{Temp_{\ desired} - Temp_{\ final}}{Temp_{\ desired} - Temp_{\ orig}} \right| \tag{11}$$

The *utility* value may vary from $(-\infty, 1.0]$ and is solely computed based on the value of the observed events. If the desired temperature is reached (e.g. Temp^1_{final} in Figure 2), the utility becomes 1.0. If the desired temperature is not reached, but has evolved closer to the desired value (e.g. Temp^2_{final} and Temp^3_{final} in Figure 2), the utility is positive but less than 1.0. If the rule made the current situation worse than it was before because the distance to the desired temperature increases rather than decreases (e.g. Temp^4_{final} in Figure 2), the utility becomes negative.

The optimality of the rule depends on the optimization criteria at hand, in this case the objective to save energy and to assure reasonable comfort levels. To

measure the impact on the energy bill, we define a monetary cost per time unit for any given state of an actuator. The comfortableness can be measured with the Predicted Mean Vote (PMV) and Predicted Percentage Dissatisfied (PPD) [15] measures. For sake of simplicity, we compute the overall optimality as a weighted average of both values, with the weights defined by the policy makers.

$$optimality = w_1 * energy\ cost + w_2 * comfort\ level \tag{12}$$

Inherently, this is a multi-criteria optimization problem. Rather than simplifying it into a single-criteria optimization problem, we have shown in previous work [17] how to tackle these using Pareto-optimization techniques.

3.3. Towards constructing a software-based safeguard for intelligent applications

The objective is to decide which action to take given a contextual state of the environment, and to ensure safe or avoid unwanted behavior. The software-based safeguard that we propose is based on the fact that we can translate rule-based adaptation into a classification problem. As input we use the contextual state and the actions taken, and where the classifier learns whether the reached outcome of rule reached its objectves (i.e. matching with the desired effects). A key concern is that commands executed by different actuators (or humans for that matter) might not be independent. We therefore need a conflict resolution strategy that can identity hidden dependencies for conflicting actions.

Our approach consolidates the different ECA rules for each actuator into a separate decision tree, where each of the nodes in the tree can be annotated with an action and corresponding effect. We assume that per actuator, only one rule can match. For example, to govern the air conditioning, we construct a decision tree for the fan speed:

$$temp: \ fanspeed \ \Rightarrow \ < off,\ low,\ medium,\ high,\ auto > \tag{13}$$

The following rules would be inconsistent due to the fact that different actions are taken for a temperature of $25°C$:

$$temp: \textbf{if}\ (temp \geq 22)\ \textbf{then}\ fanspeed = medium \tag{14}$$

$$temp: \textbf{if}\ (temp \geq 24)\ \textbf{then}\ fanspeed = high \tag{15}$$

Our decision trees are event driven, such that they are only called when the contextual state changes. However, given a particular state, there can be a single or multiple actions triggered, depending on how many actuators (i.e. decision trees) are defined. We use Hoeffding trees [18], a mining technique for high-speed data streams, to continuously test and classify adaptation rules and actions. The added value of using Hoeffding trees is manifold:

- They operate in a limited amount of memory and time.
- They are ready to predict and classify at any time.
- They can be used in an interleaved test-then-train setting.

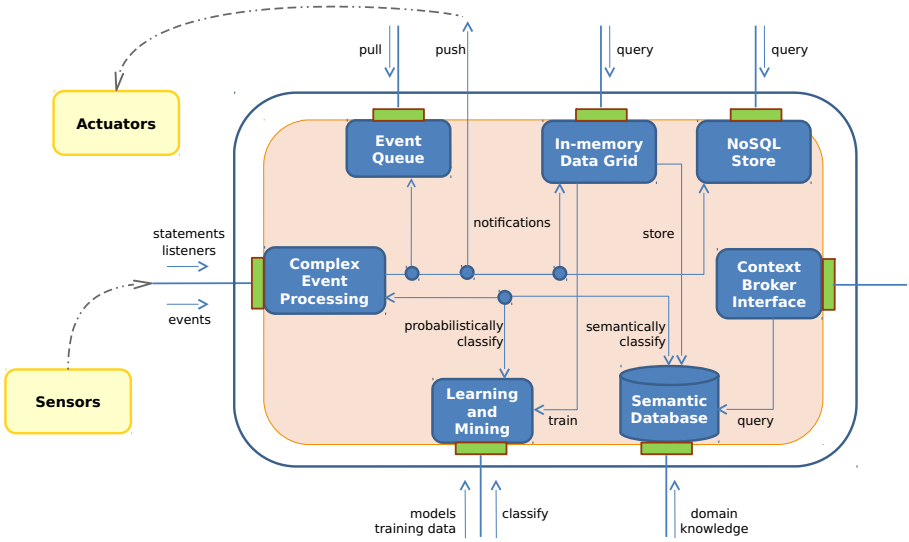

Figure 3. Proof-of-concept implementation on top of the SAMURAI architecture

During the training phase, each time a rule is triggered:

1. It will first collect the current state of all the actuators.
2. Classify the current adaptation as a positive or negative training instance depending on whether the rule was effective or not.

During the testing phase, the Hoeffding tree is used to ascertain whether the actuator adaptation will be effective or not given the state of the other actuators. Our software-based safeguard builds upon Hoeffding trees to make sure that rules with a negative utility are never executed. Additionally, the approach supports interleaved test-and-training, and is therefore suited for the problem at hand. Furthermore, these trees are well suited to adapt to changes via concept drift to handle adversarial interactions.

3.4. Implementation

For the proof-of-concept implementation, we build upon our SAMURAI[3] [19] framework, our stream mining context architecture with components for complex event processing (CEP), machine learning, knowledge representation, NoSQL persistence and in-memory data grids.

We use Esper as our CEP engine to process the incoming events and compute amongst others the PMV and PPD values. The Hoeffding decision tree implementation is provided by the Massive Online Analysis (MOA) framework [20] that is integrated into SAMURAI.

[3]https://butler.cs.kuleuven.be/samurai/

Preference	Sun Shield	Light	Airco 1 - Fan	Airco 2 - Fan	Utility
21 °C	Open	30 %	Medium	High	0.1
20 °C	Closed	70 %	Low	Medium	-2.0
23 °C	Open	20 %	High	High	0.4
19 °C	Closed	60 %	High	High	0.9
22 °C	Open	40 %	Low	High	?

Table 1. Test and training instances for a Hoeffding tree targeting *Airco 2 - Fan*

4. Evaluation

We have collected data for more than 6 months from the sensors in the ITeS building (see section 3). This gives us a reasonable idea of the sensor value ranges and the corresponding state of the actuators. Our current prototype implementation contains 23 different adaptation rules for the actuators in our case study. Table 1 provides a simplified representation of this data set and corresponding utility.

While a prototype of our approach is ready, the quantitative evaluation on a realistic setup is still work in progress. For now, to test its robustness against exceptional circumstances we rely on simulation of semi-randomized input streams, rather than creating exceptional circumstances that would require putting people through uncomfortable experiences.

In preliminary experiments with forced errors (mimicking sensor and/or actuator failures), we see the safeguard kick in by canceling out the corresponding adaptation rules. Reflecting back on the earlier questions posed in the introduction, we can evaluate from a qualitative perspective and state that:

1. Decision trees are a straightforward mechanism to detect inconsistent rule-based behavior initiated by independent contextual triggers.
2. By making the desired effect explicit in the event-condition-action rule, we can mine event streams and classify actions over time, and learn whether the rule will be effective or not.
3. Hoeffding trees are well-suited for test-then-train style classification scenarios, and they incorporate a time-based window to be able to cope with drift and to keep its model in sync with the data stream.
4. Using Hoeffding trees we can anticipate whether an action in an adaptation rule will reach its intended objective given the state of the other actuators. If the utility is negative, it will worsen the current situation and the rule will not be initiated.

Although only tested on artificial scenarios, our approach can also deal with optimization objectives, by being selective about which rule is executed given the costs and benefits that may come with it.

5. Conclusion

The overall goal of this paper was to identify relevant concepts and methodologies for the development of more reliable smart applications in intelligent environments. We have proposed stream mining solutions as a way for intelligent systems

to learn from their mistakes. Our prototype built on top of our SAMURAI supports analyzing failures in context-aware adaptation by exploiting the strengths of Hoeffding trees. This culminates in a software-based safeguard for rule-based behavior that improves the robustness of the system against unexpected events.

While a realistic lab experimental setup is up and running and data collection is ongoing, a thorough quantitative assessment with complete integration of the control feedback loop is still an ongoing work-in-progress. At the same time, we are also comparing with and evaluating the use of Dynamic Decision Networks as an alternativ approach for fault detection, identification and recovery using GeNIe and SMILE as software backends.

As the timing of observations plays a major role in fault diagnosis and event prediction, we will further investigate to what extend our approach can be extended for the diagnosis of cascading anomalies.

Acknowledgments

This research is partially funded by the Research Fund KU Leuven and the FP7 BUTLER[4] project, and conducted under the "Advanced Sensing Platform and Green Energy Application Technology Project" of the Institute for Information Industry which is subsidized by the Ministry of Economy Affairs of the Republic of China.

References

[1] K. Henricksen and R. Robinson, A Survey of Middleware for Sensor Networks: State-of-the-art and Future Directions, In *Proceedings of the International Workshop on Middleware for Sensor Networks*, MidSens'06, pp. 60–65, ACM, New York, NY, USA, 2006.

[2] C. Mascolo, L. Capra, and W. Emmerich, Mobile Computing Middleware, In *Advanced Lectures on Networking*, pp. 20–58, Springer-Verlag, New York, NY, USA, 2002.

[3] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam, Middleware for Distributed Context-aware Systems, In *Proceedings of the 2005 Confederated International Conference on On the Move to Meaningful Internet Systems - Volume Part I*, OTM'05, pp. 846–863, Springer-Verlag, Berlin, 2005.

[4] D. C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[5] E. Kim, S. Helal, and D. Cook, Human Activity Recognition and Pattern Discovery, *IEEE Pervasive Computing*, **9**, 1, (2010), 48–53.

[6] W. van der Aalst, T. Weijters, and L. Maruster, Workflow Mining: Discovering Process Models from Event Logs, *IEEE Trans. on Knowl. and Data Eng.*, **16**, 9, (2004), 1128–1142.

[7] W. Yao, C.-H. Chu, and Z. Li, Leveraging Complex Event Processing for Smart Hospitals Using RFID, *J. Netw. Comput. Appl.*, **34**, 3, (2011), 799–810.

[8] M. Buettner, R. Prasad, M. Philipose, and D. Wetherall, Recognizing Daily Activities with RFID-based Sensors, In *Proceedings of the 11th International Conference on Ubiquitous Computing*, Ubicomp'09, pp. 51–60, ACM, New York, NY, USA, 2009.

[9] A. Aztiria, J. C. Augusto, R. Basagoiti, A. Izaguirre, and D. J. Cook, Discovering Frequent User–environment Interactions in Intelligent Environments, *Personal Ubiquitous Comput.*, **16**, 1, (2012), 91–103.

[4] http://www.iot-butler.eu

[10] R. Cook, How Complex Systems Fail, Cognitive technologies Laboratory, University of Chicago. http://www.ctlab.org/documents/How Complex Systems Fail.pdf, 2000.
[11] J. C. Augusto and M. J. Hornos, Software simulation and verification to increase the reliability of Intelligent Environments, *Advances in Engineering Software*, **58**, (2013), 18–34.
[12] J. C. Augusto, P. J. McCullagh, and J.-A. Augusto-Walkden, Living without a safety net in an intelligent environment, *EAI Endorsed Transactions on Ambient Systems*, **11**, 10-12, (2011).
[13] A. Schuster, *Robust Intelligent Systems*, Springer Publishing Company, Incorporated, 1 edition, 2008.
[14] B. Lindsay, Designing for failure may be the key to success. - interview by steve bourne, *ACM Queue*, **2**, 8, November 2004.
[15] ISO EN 7730-2005, Ergonomics of the thermal environment - analytical determination and interpretation of thermal comfort using calculation of the pmv and ppd indices and local thermal comfort criteria, Technical report, 2005.
[16] D. Preuveneers and Y. Berbers, Consistency in Context-Aware Behavior: a Model Checking Approach, In *Workshop Proceedings of the 8th International Conference on Intelligent Environments. Ambient Intelligence and Smart Environments*, volume 13, pp. 401–412, IOS Press, 2012.
[17] A. kishore Ramakrishnan, N. Z. Naqvi, Z. W. Bhatti, D. Preuveneers, and Y. Berbers, Learning deployment trade-offs for self-optimization of internet of things applications, In *Proceedings of the 10th International Conference on Autonomic Computing, ICAC'13*, pp. 213–224, USENIX, San Jose, CA, 2013.
[18] P. Domingos and G. Hulten, Mining high-speed data streams, In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'00, pp. 71–80, ACM, New York, NY, USA, 2000.
[19] D. Preuveneers and Y. Berbers, SAMURAI: A Streaming Multi-tenant Context-Management Architecture for Intelligent and Scalable Internet of Things Applications, In *Proceedings of the 10th International Conference on Intelligent Environments*, 2014 (to appear).
[20] A. Bifet, J. Read, B. Pfahringer, G. Holmes, and I. Zliobaite, CD-MOA: Change detection framework for massive online analysis, *Lecture Notes in Computer Science*, **8207**, (2013), 92–103.